ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 4, April 2016

Reconfigurable Kernel for Cortex M3-1788 ARM

Rajamohan.L[1], Dr.Ravi.S[2]

Research Scholar, Department of ECE, St.Peter's University, Chennai, India[1]

Professor and Head, Department of ECE, Dr.M.G.R. Educational and Research Institute University, Madhuravoyal, Chennai, India[2]

ABSTRACT : Major issues in development of Embedded Systems include protecting available platforms to develop portable applications with optimal form factor, improving software design processes, providing reconfigurable kernels to manage IO resources and memory, etc. Traditional operating systems are too bulky and lack modularity and pluggable features. Reconfigurability can be achieved at architecture level or program level and involves the use of field programmable gate arrays (FPGAs), multiple ARM cores, Programmable kernel and user space options etc. In this paper, the design techniques customize a computational fabric for specific applications. The paper proposes reconfigurable process in Linux wherein after initially the target is bootloaded the basic commands that are used to execute and load files and handling of storage space is done dynamically.

KEYWORDS: Reconfigurable kernel, reconfigurable user space, bootloader, ARM core

I. INTRODUCTION

Embedded Linux is a small operating system (approximately100Kb size) which comes from the kernel cutting and optimizing of standard Linux. It is essential to maintain the kernel size optimal even when coupled with other necessary modules and applications. A typical embedded system consists of microprocessor, memory, network capability, intrinsic physical layer error detection schemes, ports (like USB/serial/parallel). Typical OS like Linux is based on an open-source and is popular in target embedded system, particularly ARM cores. Typical realtime appliances using embedded targets with linux kernel focus on such issues as boot-up time, power management, real-time and kernel size. To make fast boot-up time in the Embedded Linux, it is necessary to package root file-system through the configuration analysis of bootloader and different root file-system. In this paper, the focus is on optimal usage of kernel size and to make the target hardware reconfigurable with respect to kernel functions, device driver functions, user space functions and advanced user space functions. At every level, the memory utilization is monitored dynamically and includes the incremental variation, inode address location mapped etc.

To reduce the energy consumption the number of realized reconfigurations is kept dynamic using an external storage and offer power efficient, compact designs. The reconfiguration exposes the system to events that may or may not be known at design time and the programmable hardware at runtime allow alternative execution of various kernels that could conserve space in embedded systems, and thereby providing a balance between performance and area. The reconfiguration of the target can be implemented on a static or a dynamic level. In static reconfiguration (often referred to the compile time reconfiguration) hardware resources remain static for the life of the design and are possible only on the Register Transfer Level (RTL) development stage. The dynamic reconfiguration (often referred to the runtime reconfiguration) uses a dynamic allocation scheme that reallocates hardware at runtime. It can increase the system performance using highly optimized circuits that are loaded and unloaded or reconfigured dynamically during the operation of the system. The main design issues to be handled with the system modeling of the dynamic reconfigurable systems includes:

1) validation of reconfigurable systems behavior during the functioning and reconfiguration
2) evaluation of the task execution time and resource utilization in a reconfigurable system;
3) preliminary evaluation of energy consumption during functionality and reconfiguration;

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 4, April 2016

II.RELATED WORKS

Ci Wenyan et. al.(2010) described the methods and skills on transplanting Linux to ARM S3C2410 microprocessor-based target board. A set of pragmatic method of Linux transplantation has been introduced specifically for small embedded platform, including building cross-compile environment, compiling Bootloader, amending, configuring and compiling Linux kernel and producing cramfs file system and so on. After transplantation of Linux kernel, its size can be controlled in 700KB.

Elena Suvorova et. al. (2015) proposed in his paper, methods of dynamically reconfigurable multi-core System-on-chip (SoC) design, and also presented the approaches of system modeling for evaluation of these systems. The dynamically reconfigurable SoC can be developed using the FPGA and the ASIC technologies. In this paper the existing tools for SoC system design and the requirements for it to allow modeling of reconfigurable systems are considered.

A.Shalimov(2014) presented the novel approach on offloading the most time consuming and frequently used functionality of the SDN/OpenFlow controller to the Linux kernel space. This speeds up network applications in 2.5 times together with possibility of using the all userspace libraries and programming tools. SDN/Openflow is already a mainstream in the area of computer networks. It allows us to automate and to simplify network management and administration.

Like Yan et. al.(2010), proposed a Reconfigurable Multi- Core (RMC) architecture combining general multi-core and reconfigurable logic. The Reconfigurable Logic is logically divided into Reconfigurable Processing Units (RPUs), which are coupled with General Purpose Cores (GPCs) as coprocessors via a configurable full crossbar switch. And a RPUManager (RPU-M) is designed to manage the RPUs.

III .IMPLEMENTATION

3.1 Boot-up Process

The boot-up process depends on the hardware platform being used. However, once the kernel is identified and loaded by the bootloader, the default boot-up process is identical across all architectures. When the system starts up, it launches the first stage machine code instructions on the MBR (Master Boot record) of the flash memory, called a bootloader. The Bootloader loads itself into memory and it yields control of the boot-up process to it. Also, the bootloader is mainly responsible for loading the kernel and the root file-system. It mainly tests or initializes hardware system. Types of bootloader vary greatly between architectures and is shown in Table 1.

Table 1. Types of Bootloader

Processor	BootLoader	Characteristics
X86	LILO	The main disk bootloader from Linux
	GRUB	GNU's successor of LILO
ARM	BLOB	Loader from LART hardware project
	RedBoot	eCos- based loader
MPC	PPC Boot	Supports various platforms based on MPC
	U- boot	Universal Loader based on PPC

The second stage bootloader loads the compressed kernel image into memory at the time of boot-up, and inturn loads any necessary modules in a decompressed state. The kernel sets the environment required for a system and manages the various resources. It also communicates with hardware through the use of interrupt and provides services through the use of System Call interface. Subsequently, the bootloader places the appropriate root file system image into memory, and mounts the root partition read-only. The root file-system is used by the kernel to load drivers necessary to boot-up the system. Once the kernel and the root file-system image are loaded into memory, the bootloader gives control of the boot-up process to the kernel. The u-boot utility is a multi-platform, open-source, universal boot-loader with comprehensive support for loading and managing boot images, such as the Linux kernel.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 4, April 2016

3.2 Steps to Boot the board

(1)Board should be preloaded with u-boot (2) Insert sd card into the board (3) During u-boot countdown press any key to stop the boot (4) Type "fat32load"(5) Type bootm (6) Wait until Linux shell prompt appear (7) Type "mount /dev/mmcblk0p1 /mnt (8) cp /mnt/program / (9) type ./program to run the reconfig program.

3.3 The Root file-system

The different root file systems along with their characteristics is presented in table 2. Some support uncompressed format (ex:CRAMFS), while some can be executed in place (Ex:XIP)

Table 2. File system characteristics

Filesystem	Write	Persistent	Power down reliability	Compression	Lives in RAM
CRAMFS	No	N/A	N/A	Yes	No
JFFS2	Yes	Yes	Yes	Yes	No
JFFS	Yes	Yes	Yes	No	No
Ext2 over NFTL	Yes	Yes	No	No	No
Ext3 over NFTL	Yes	Yes	Yes	No	No
Ext2 over RAM disk	Yes	No	No	No	Yes

IV. HARDWARE IMPLEMENTATION

Table 3. The dynamic functions

	Modules		
classification	**Kernel Space**	**User Space**	**Advanced user space**
	CramFs support	Tar command	lsusb
	I2c support	Cal command	Find command
	USB mass storage support	Chmod command	Fdisk command
	USB to serial converter support	Rmdir command	
	Unix domain sockets support	Sort command	

Fig.1 ARM core with root file system and dynamic kernel mounted in Sd-card

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 4, April 2016

1-> Target ARM core with 'u-boot' bootloader and kernel image
2-> Hyperterminal showing the results.

V. RESULTS AND DISCUSSION

In this work, test case functionality provides facility to test and execute basic commands. Initially there are three modules - kernel space, user space and advanced commands. Each of these modules has its special functions intrinsically. The main purpose of this test case functionality is to include each and every command individually before executing. If executed before adding the commands then, it will not recognize. Also, once each command is added to kernel, it recognizes the command in the shell and also tracks the memory space that gets added due to the included command. The implementation results are shown in figure 2 to figure 8 for inclusion/deletion of different modules of kernel space, user space and advanced user space modules.

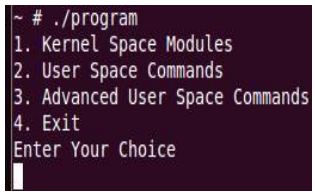

Fig 2. Listing of main modules

In fig 2, different kinds of modules such as kernel space modules, user space commands and advanced user space commands are listed so that any one of the module is selected for further processing.

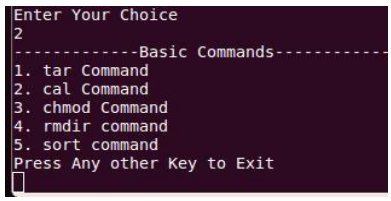

Fig 3. Listing submodules included in user space (listed in table 3)

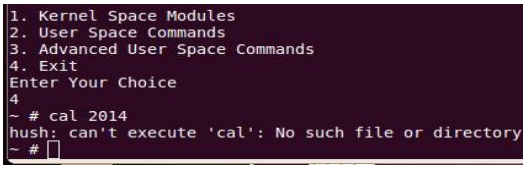

Fig 4. Target kernel not recognizing 'cal' command initially

Fig. 4 displays the initial execution of cal command. It shows an error message as no such file found since cal command is not added to the basic command.

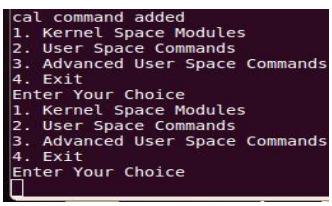

Fig 5. cal command added

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 4, April 2016

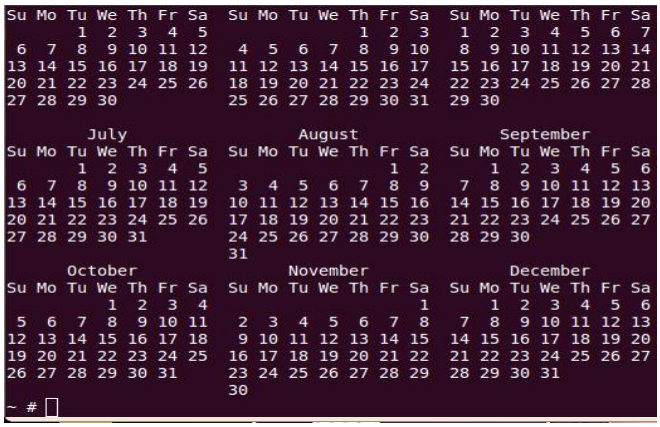

Fig 6. Execution of cal command after inclusion

Fig 5 & 6 displays the inclusion of cal command and its execution.

Fig 7. 'Unix domain socket command' included

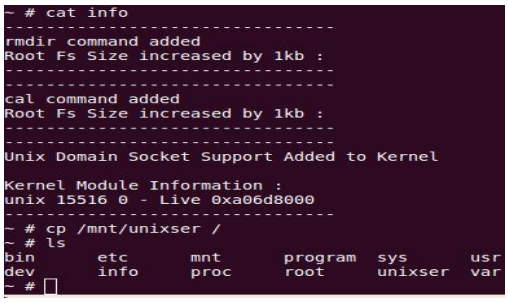

Fig 8. Displaying kernel Size and other relevant details after inclusion of different shell commands like rmdir, cal, unix domain socket and unixser

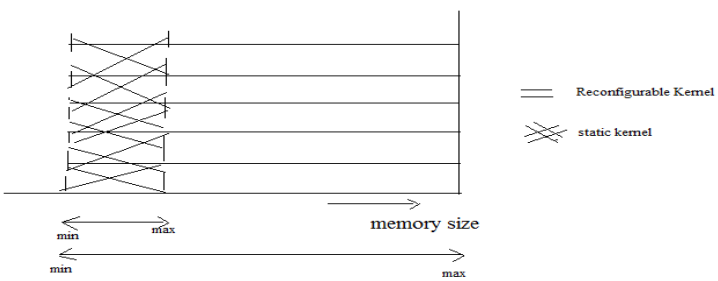

Fig.9 memory utilization for static and reconfigurable kernel

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 4, April 2016

VI. CONCLUSION

A target that contains one or more processors and a reconfigurable fabric upon which custom functional units is developed is presented. The processor(s) executes sequential and noncritical code, and in runtime additional codes can be efficiently mapped to hardware and executed by processing units that have been mapped to the reconfigurable fabric. Static kernel has fixed memory sizes by which lot of memory spaces is not efficiently used. In reconfigurable kernel, commands are included and executed when needed which reduces memory size and occupies less memory space.

REFERENCES

[1]Ashish Srivatsava, " Multiple kernel learning for sparse representation based classification", IEEE Transaction on Image processing, Vol.23, No. 7, pp 3013 - 3024, July 2014.
[2] Carsten Boke, "Reconfigurable real time operating systems and their applications, IEEE 2003.
[3] Ci Wenyan, "Methods and skills on transplanting Linux to ARM S3C2410, IEEE 2010.
[4] Elena Suvarova, " System level modeling of Dynamic Reconfigurable System on chip", Proceeding of 17th conference of Fruct association, 2015.
[5] Jianfeng Zhu, " A hybrid reconfigurable architecture and design methods aiming at control intensive kernels", IEEE Transactions on VLSI systems, pp 1-10,2014..
[6] Jongmoo Choi, " Kernel aware module verification for robust reconfigurable operating system,"Journal of Information science and Engineering, pp 1339 -1347, 2007.
[7] Katherine Compton, Scott Hauck, " Reconfigurable computing : A survey of systems and software", ACM Computing surveys, Vol 34, No 2, pp 171- 210, June 2007.
[8] Kyung ho Chung, " A study on the packaging for fast boot-up time in Embedded Linux", IEEE 2007.
[9] Leonid veytser, " A Linux Kernel implementation of broadcast interflow network coding", pp 1732 - 1738, IEEE 2013.
[10] Like Yan, " A reconfigurable processor architecture combining multi-core and reconfigurable processing unit", 10th International conference on computer and Information Technology, pp 2897 - 2902 , 2010.
[11] Liu Guangzhong, " Research and analysis on Reconfigurable system", Proceedings of International multi conference of Engineers and Computer scientists, Vol 1, 2008.
[12] Russell Tessier, " Reconfigurable Computing Architectures", Proceedings of IEEE, Vol. 103, No.3, pp 332- 354, Mar 2015.
[13] Scott Hauck, "The Future of Reconfigurable Systems", Keynote Address, 5th Canadian Conference on Field Programmable devices, June 1998.
[14] A.Shalimov," In-Kernel offloading of an SDN/OpenFlow controller", IEEE 2014.
[15] Xin Xu, " Kernel based approximate dynamic programming for real time online learning control: An experimental study", IEEE Transactions on control system technology, Vol 22, No.1 , Jan 2014, pp - 146 - 156

APPENDIX

Build Cross-Compile

Cross-compiling environment is determined by the compiler, linker and interpreter composed of an integrated development environment.

The compile process is as follows:

A. First, download pre-compiled cross development environment to / directory

B. Execute following instructions to install:

1) #cd / 2) #tar -zxvf arm-linux-gcc-2.95.3.tgz

C. The compile tool produced is in directory /usr/local/arm/3.4.1/bin, amend /etc/profile.

Add the following line: pathmunge /usr/local/arm/2.95.3/bin So we can run arm-linux-gcc directly without writing the absolute path. Then restart.

D. type the following instruction finally: #arm-linux-gcc –v

Tar Command

The Linux "**tar**" stands for tape archive, which is used by large number of **Linux/Unix** system administrators to deal with tape drives backup. The tar is most widely used command to create compressed archive files and that can be moved easily from one disk to anther disk or machine to machine.

Basic Commands

1. cd /
2. mkdir test
3. cd test
4. echo 'hello' >> sample
5. cd /
6. tar cvf testtar.tgz test

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 4, April 2016

7. rm –r test
8. tar –xvf testtar.tgz

fdisk command : fdisk [*options*] [*device*]

System administration command. **fdisk** displays information about disk partitions, creates and deletes disk partitions, and changes the active partition. The minimum recommended size for a Linux system partition is 40 MB. Normally, each *device* will be */dev/hda*, */dev/hdb*, */dev/sda*, */dev/sdb*, */dev/hdc*, */dev/hdd*, and so on. An interactive, menu-driven mode is also available.

1. Insert USB pendrive to the board
2. Enable USB mass storage functionality in reconfig program
3. Enable fdisk command from the reconfig program
4. fdisk /dev/sda1
5. type 'quit' to exit fdisk prompt.

Test Kernel functionality

Cramfs

1. cp /mnt/cramfs.img /
2. Enable cramfs functionality from the reconfig program
3. Type "losetup /dev/loop0 cramfs.img"
4. mkdir /cramtest
5. mount –t cramfs /dev/loop0 /cramtest
6. cd cramtest
7. ls
8.

i2ctest

1. Enable i2c functionality from the reconfig program
2. cp /mnt/i2ctest /
3. ./i2ctes

USBStorage

1. Enable USB Mass Storage Functionality from the reconfig program
2. mkdir /usb
3. mount /dev/sda1 /usb
4. cd /usb
5. ls

USb to Serial Converter

1. Enable USB to Serial Functionality from the reconfig program
2. cat /dev/ttyUSB0

Unix Domain Sockets

1. Enable Unix Domain Sockets Functionality from the reconfig program
2. cp /mnt/unixser /
3. ./unixser